eda-done

December 8, 2022

[1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
import os
```

[2]:
```
from typing import Optional, Dict, List
import tensorflow as tf
import os
```

[3]:
```
DATA_DIR = "/data/images/"
```

[4]:
```
#current folder path
IMAGE_PATH = os.getcwd() + DATA_DIR
```

[5]:
```
class ImagePot:
    # parsed from abbreviations.csv
    CLASSNAME_LOOKUP = {
        'ABE': 'Abnormal eosinophil',
        'ART': 'Artefact',
        'BAS': 'Basophil',
        'BLA': 'Blast',
        'EBO': 'Erythroblast',
        'EOS': 'Eosinophil',
        'FGC': 'Faggott cell',
        'HAC': 'Hairy cell',
        'KSC': 'Smudge cell',
        'LYI': 'Immature lymphocyte',
        'LYT': 'Lymphocyte',
        'MMZ': 'Metamyelocyte',
        'MON': 'Monocyte',
        'MYB': 'Myelocyte',
        'NGB': 'Band neutrophil',
        'NGS': 'Segmented neutrophil',
        'NIF': 'Not identifiable',
        'OTH': 'Other cell',
```

```
        'PEB': 'Proerythroblast',
        'PLM': 'Plasma cell',
        'PMO': 'Promyelocyte',
    }

    def __init__(self, image_paths: List[str], classname: str, encoding:␣
↪Optional[List[int]] = None):
        self.image_paths = image_paths
        self.classname = classname
        self.encoding = encoding
        self.full_name = self.CLASSNAME_LOOKUP.get(classname, "?")

    def set_encoding(self, encoding) -> None:
        self.encoding = encoding

    def __str__(self) -> str:
        return "Image Pot containing Class {0} ({1}), {2} file(s)".format(
            self.classname,
            self.full_name,
            len(self.image_paths),
        )

    def __repr__(self) -> str:
        return str(self)
```

[6]:
```
def load_images(
        num_image_minimum: int = 100,
        num_image_limit: Optional[int] = 200,
        print_files_on_disk: bool = False,
    ) -> Dict[str, Dict[str, object]]:

    # these are the labels enclosing each folder of images
    sub_dirs = sorted([y for x in os.walk(IMAGE_PATH) if (y := (x[0].
↪split(IMAGE_PATH)[-1]))])

    # assume that data processing script has already un-nested image contents
    good_folders = {}

    for folder in sub_dirs:
        file_names = os.listdir("/".join([IMAGE_PATH, folder]))
        num_images = len(file_names)
        if print_files_on_disk:
            print(f"Classname: {folder} has {num_images} images.")

        if num_images >= num_image_minimum:
            good_folders[folder] = {
```

```
                'pot': ImagePot([IMAGE_PATH + folder + "/" + x for x in␣
↪file_names[:num_image_limit]], folder),
                'classname': folder,
                'ohe': None,
            }

    # directory of valid images
    ohe_classes = [([0] * len(good_folders)) for _ in range(len(good_folders))]
    for i, k in enumerate(good_folders):
        encoding = ohe_classes[i]
        # set equal to available classes
        encoding[i] = 1
        good_folders[k]['pot'].set_encoding(encoding)

        # simplify repo structure
        good_folders[k] = good_folders[k]['pot']

    return good_folders
```

[7]:
```
data_repo = load_images()
```

[8]:
```
data_repo
```

[8]:
```
{'ART': Image Pot containing Class ART (Artefact), 200 file(s),
 'BAS': Image Pot containing Class BAS (Basophil), 200 file(s),
 'BLA': Image Pot containing Class BLA (Blast), 200 file(s),
 'EBO': Image Pot containing Class EBO (Erythroblast), 200 file(s),
 'EOS': Image Pot containing Class EOS (Eosinophil), 200 file(s),
 'HAC': Image Pot containing Class HAC (Hairy cell), 200 file(s),
 'LYT': Image Pot containing Class LYT (Lymphocyte), 200 file(s),
 'MMZ': Image Pot containing Class MMZ (Metamyelocyte), 200 file(s),
 'MON': Image Pot containing Class MON (Monocyte), 200 file(s),
 'MYB': Image Pot containing Class MYB (Myelocyte), 200 file(s),
 'NGB': Image Pot containing Class NGB (Band neutrophil), 200 file(s),
 'NGS': Image Pot containing Class NGS (Segmented neutrophil), 200 file(s),
 'NIF': Image Pot containing Class NIF (Not identifiable), 200 file(s),
 'OTH': Image Pot containing Class OTH (Other cell), 200 file(s),
 'PEB': Image Pot containing Class PEB (Proerythroblast), 200 file(s),
 'PLM': Image Pot containing Class PLM (Plasma cell), 200 file(s),
 'PMO': Image Pot containing Class PMO (Promyelocyte), 200 file(s)}
```

0.0.1 Visualizing Pathology Samples

Messing with the distribution of RGB values.

[9]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
import random
```

[10]:
```
folder_list = list(data_repo.keys())
image_pot_list = []
def image_pots(data_repo, folder_list):
    for i in folder_list:
        image_pot_list.append((str(i)+'_image_pot'))
    return image_pot_list
```

[11]:
```
#function to get a list of images from the pathology folder
def get_images(image_pot):
    image_list = []
    for i in range(len(art_image_pot.image_paths)):
        image_list.append(cv2.imread(image_pot.image_paths[i]))

    return image_list
```

[12]:
```
#pulling 10 random samples for visualization
def rand_img(image_list):
    sample_images = []
    num_list = random.sample(range(0, 200), 7)
    for i in num_list:
        sample_images.append(image_list[i])
    return sample_images, num_list
```

ART

[13]:
```
art_image_pot = data_repo['ART']
ART_images = get_images(art_image_pot)
ART_sample_images, ART_sample_indices = rand_img(ART_images)
```

BAS

[14]:
```
BAS_image_pot = data_repo['BAS']
BAS_images = get_images(BAS_image_pot)
BAS_sample_images, BAS_sample_indices = rand_img(BAS_images)
```

BLA

[15]:
```
BLA_image_pot = data_repo['BLA']
BLA_images = get_images(BLA_image_pot)
BLA_sample_images, BLA_sample_indices = rand_img(BLA_images)
```

EBO

[16]:
```
EBO_image_pot = data_repo['EBO']
EBO_images = get_images(EBO_image_pot)
EBO_sample_images, EBO_sample_indices = rand_img(EBO_images)
```

EOS

[17]:
```
EOS_image_pot = data_repo['EOS']
EOS_images = get_images(EOS_image_pot)
EOS_sample_images, EOS_sample_indices = rand_img(EOS_images)
```

HAC

[18]:
```
HAC_image_pot = data_repo['HAC']
HAC_images = get_images(HAC_image_pot)
HAC_sample_images, HAC_sample_indices = rand_img(HAC_images)
```

LYT

[19]:
```
LYT_image_pot = data_repo['LYT']
LYT_images = get_images(LYT_image_pot)
LYT_sample_images, LYT_sample_indices = rand_img(LYT_images)
```

MMZ

[20]:
```
MMZ_image_pot = data_repo['MMZ']
MMZ_images = get_images(MMZ_image_pot)
MMZ_sample_images, MMZ_sample_indices = rand_img(MMZ_images)
```

MON

[21]:
```
MON_image_pot = data_repo['MON']
MON_images = get_images(MON_image_pot)
MON_sample_images, MON_sample_indices = rand_img(MON_images)
```

MYB

[22]:
```
MYB_image_pot = data_repo['MYB']
MYB_images = get_images(MYB_image_pot)
MYB_sample_images, MYB_sample_indices = rand_img(MYB_images)
```

NGB

[23]:
```
NGB_image_pot = data_repo['NGB']
NGB_images = get_images(NGB_image_pot)
NGB_sample_images, NGB_sample_indices = rand_img(NGB_images)
```

NGS

[24]:
```
NGS_image_pot = data_repo['NGS']
NGS_images = get_images(NGS_image_pot)
NGS_sample_images, NGS_sample_indices = rand_img(NGS_images)
```

NIF

[25]:
```
NIF_image_pot = data_repo['NIF']
NIF_images = get_images(NIF_image_pot)
NIF_sample_images, NIF_sample_indices = rand_img(NIF_images)
```

OTH

[26]:
```
OTH_image_pot = data_repo['OTH']
OTH_images = get_images(OTH_image_pot)
OTH_sample_images, OTH_sample_indices = rand_img(OTH_images)
```

PEB

[27]:
```
PEB_image_pot = data_repo['PEB']
PEB_images = get_images(PEB_image_pot)
PEB_sample_images, PEB_sample_indices = rand_img(PEB_images)
```

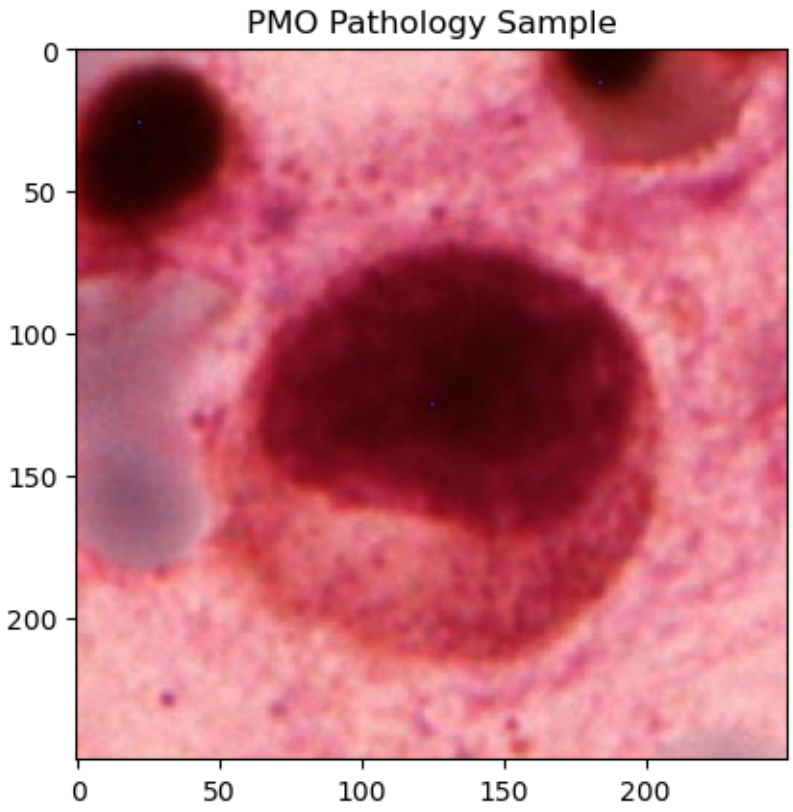
PLM

[28]:
```
PLM_image_pot = data_repo['PLM']
PLM_images = get_images(PLM_image_pot)
PLM_sample_images, PLM_sample_indices = rand_img(PLM_images)
```

PMO

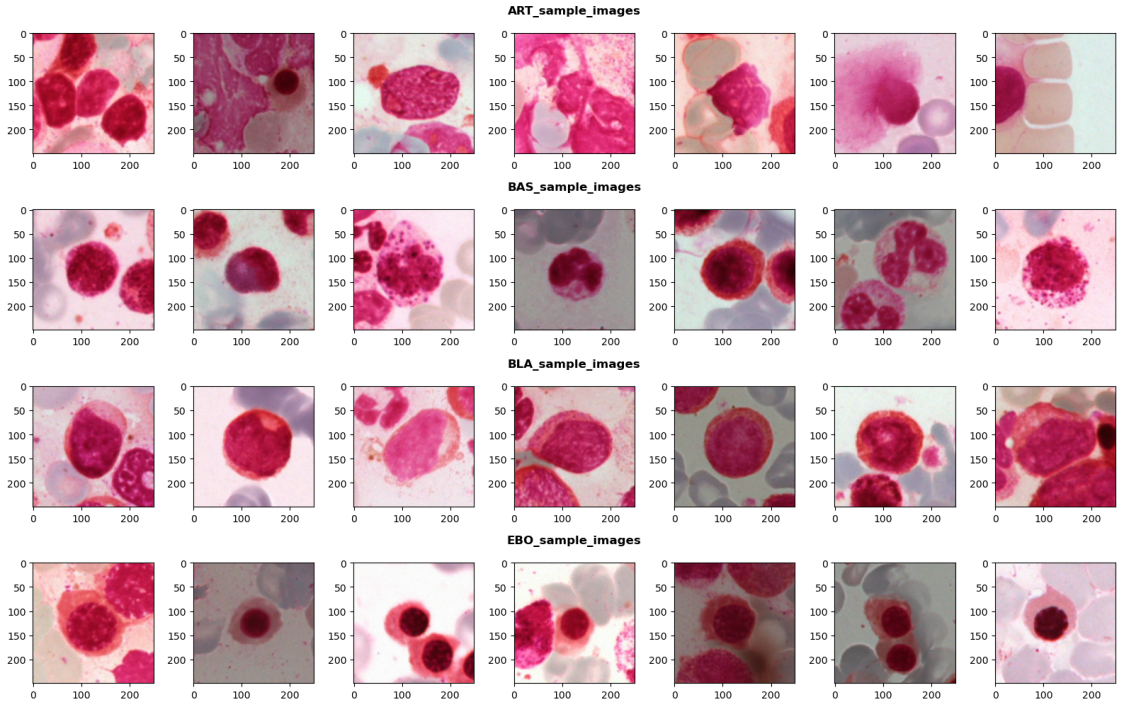
[29]:
```
PMO_image_pot = data_repo['PMO']
PMO_images = get_images(PMO_image_pot)
PMO_sample_images, PMO_sample_indices = rand_img(PMO_images)
```

[30]:
```
#testing an image
img = PMO_sample_images[3]

plt.title('PMO Pathology Sample')
imgplot = plt.imshow(img)
```



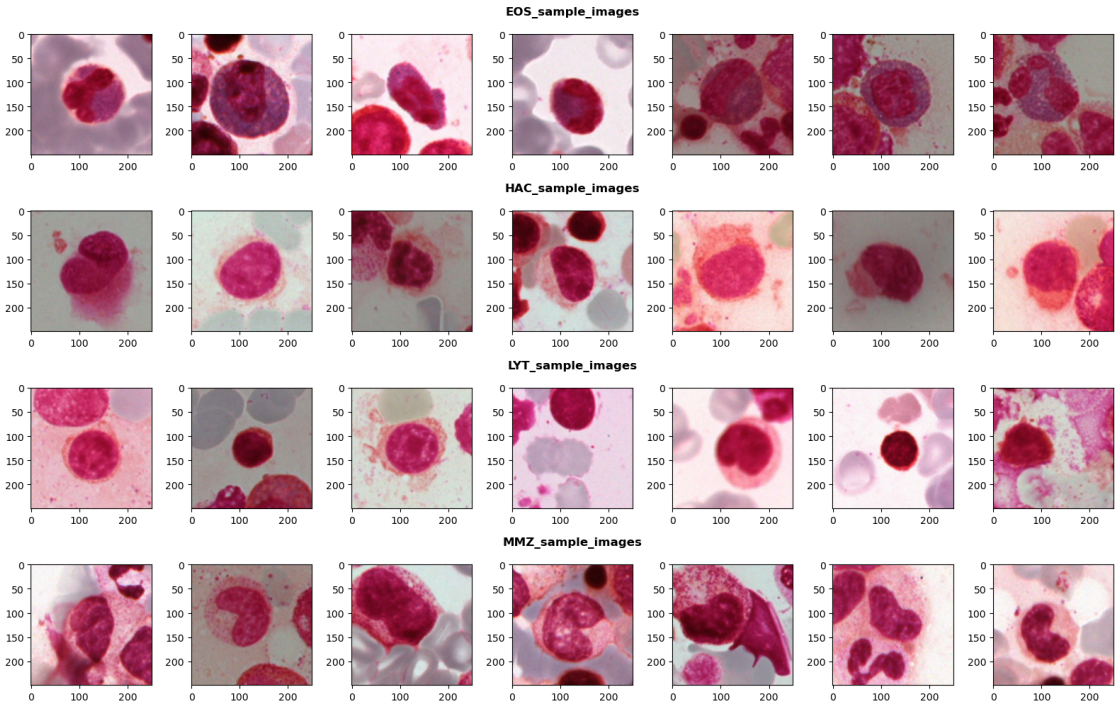
[31]:
```
#visualizing 7 samples of the first 4 pathogens
import matplotlib as mpl
from matplotlib.gridspec import SubplotSpec

def create_subtitle(fig: plt.Figure, grid: SubplotSpec, title: str):
    "Sign sets of subplots with title"
    row = fig.add_subplot(grid)
    # the '\n' is important
    row.set_title(f'{title}\n', fontweight='semibold')
    # hide subplot
    row.set_frame_on(False)
    row.axis('off')

fig, axs = plt.subplots(4, 7)
fig.set_size_inches(16, 10)

for i in range(7):
    axs[0][i].imshow(ART_sample_images[i])
    axs[1][i].imshow(BAS_sample_images[i])
    axs[2][i].imshow(BLA_sample_images[i])
    axs[3][i].imshow(EBO_sample_images[i])
```

```
rows = 4
cols = 7
grid = plt.GridSpec(rows, cols)
create_subtitle(fig, grid[0, ::], 'ART_sample_images')
create_subtitle(fig, grid[1, ::], 'BAS_sample_images')
create_subtitle(fig, grid[2, ::], 'BLA_sample_images')
create_subtitle(fig, grid[3, ::], 'EBO_sample_images')

plt.tight_layout(h_pad=2)
```



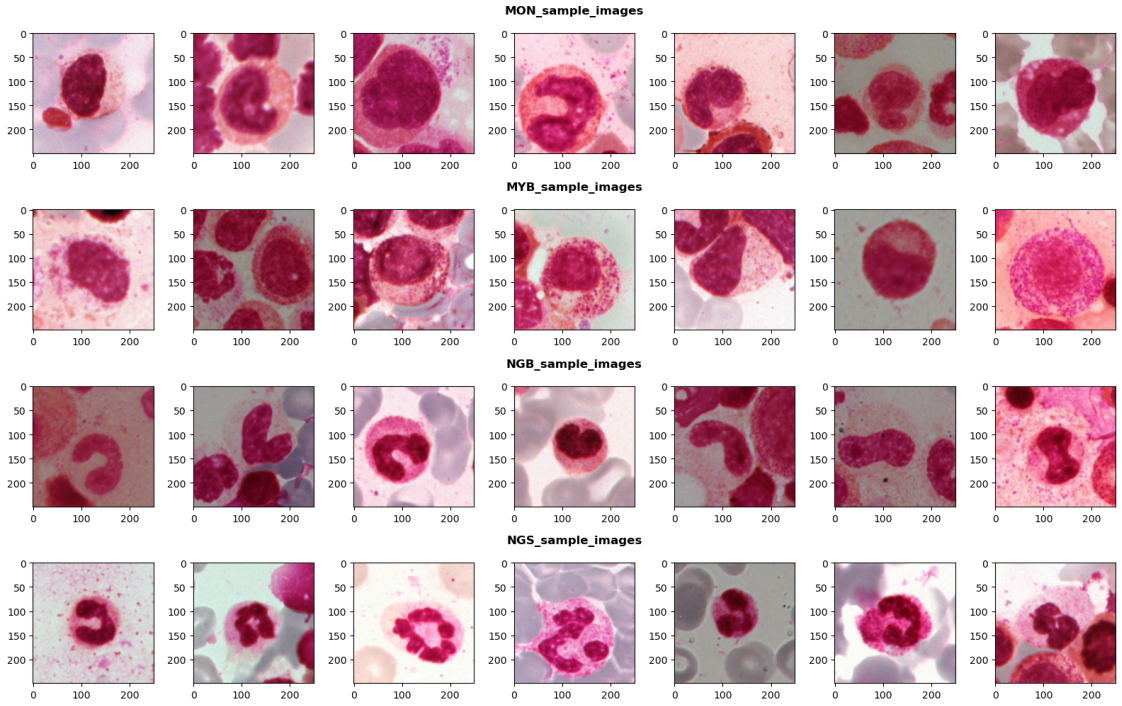
[32]:
```
#visualizing the next 4 pathogens
fig, axs = plt.subplots(4, 7)
fig.set_size_inches(16, 10)

for i in range(7):
    axs[0][i].imshow(EOS_sample_images[i])
    axs[1][i].imshow(HAC_sample_images[i])
    axs[2][i].imshow(LYT_sample_images[i])
    axs[3][i].imshow(MMZ_sample_images[i])

rows = 4
cols = 7
grid = plt.GridSpec(rows, cols)
```

```
create_subtitle(fig, grid[0, ::], 'EOS_sample_images')
create_subtitle(fig, grid[1, ::], 'HAC_sample_images')
create_subtitle(fig, grid[2, ::], 'LYT_sample_images')
create_subtitle(fig, grid[3, ::], 'MMZ_sample_images')

plt.tight_layout(h_pad=2)
```

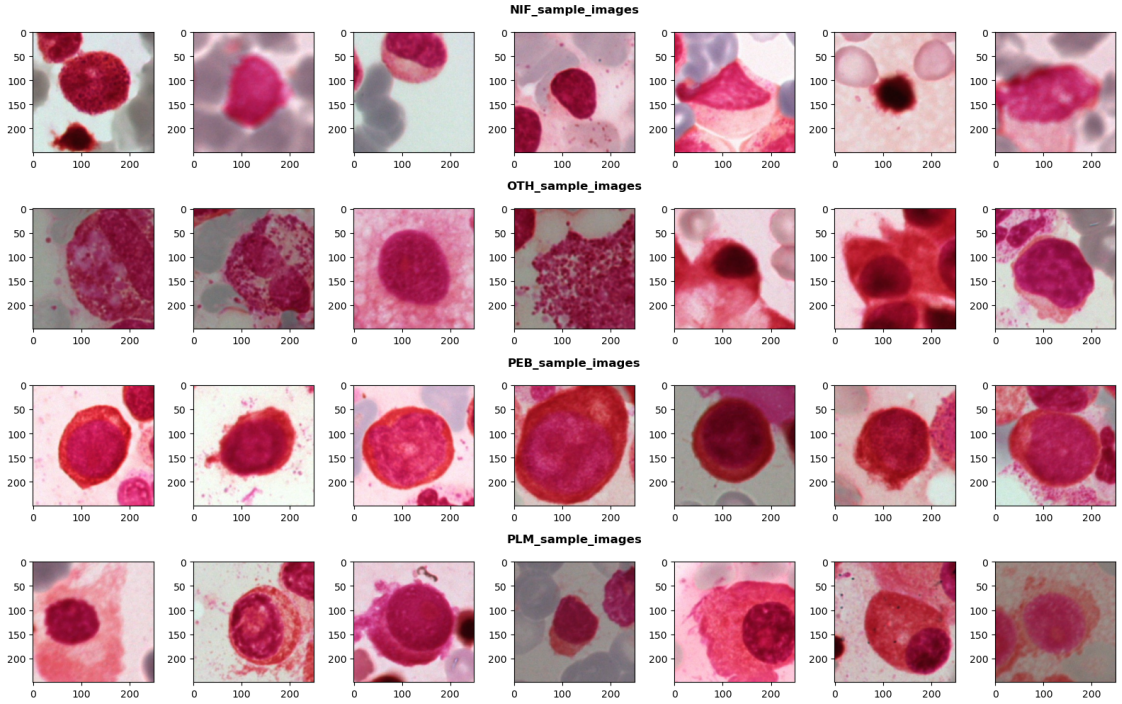


[33]:
```
#visualizing the next 4 pathogens
fig, axs = plt.subplots(4, 7)
fig.set_size_inches(16, 10)

for i in range(7):
    axs[0][i].imshow(MON_sample_images[i])
    axs[1][i].imshow(MYB_sample_images[i])
    axs[2][i].imshow(NGB_sample_images[i])
    axs[3][i].imshow(NGS_sample_images[i])

rows = 4
cols = 7
grid = plt.GridSpec(rows, cols)
create_subtitle(fig, grid[0, ::], 'MON_sample_images')
create_subtitle(fig, grid[1, ::], 'MYB_sample_images')
create_subtitle(fig, grid[2, ::], 'NGB_sample_images')
create_subtitle(fig, grid[3, ::], 'NGS_sample_images')
```

```
plt.tight_layout(h_pad=2)
```

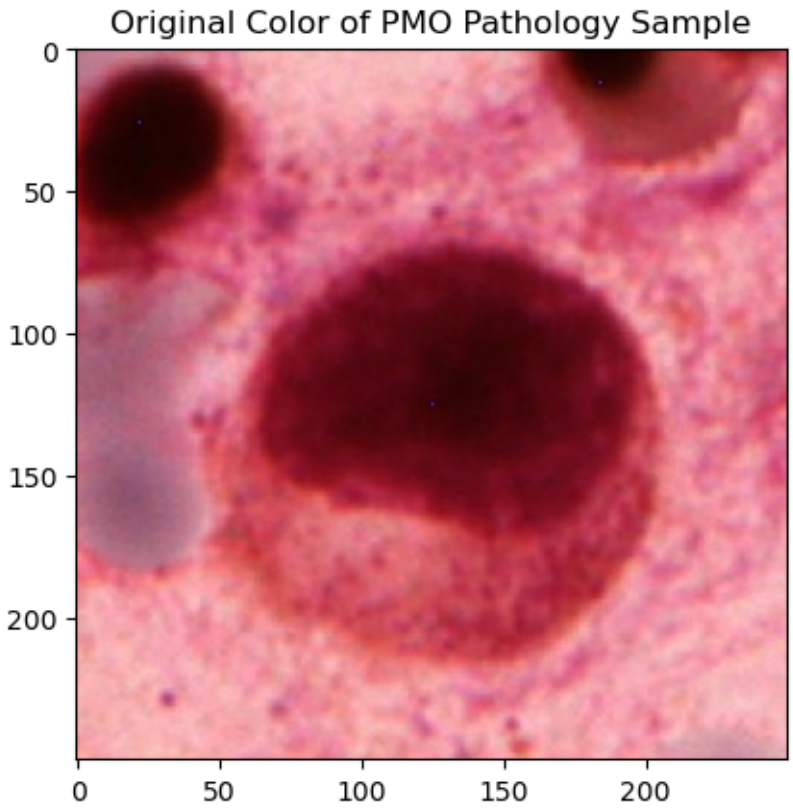


[34]:
```
#visualizing the next 4 pathogens
fig, axs = plt.subplots(4, 7)
fig.set_size_inches(16, 10)

for i in range(7):
    axs[0][i].imshow(NIF_sample_images[i])
    axs[1][i].imshow(OTH_sample_images[i])
    axs[2][i].imshow(PEB_sample_images[i])
    axs[3][i].imshow(PLM_sample_images[i])

rows = 4
cols = 7
grid = plt.GridSpec(rows, cols)
create_subtitle(fig, grid[0, ::], 'NIF_sample_images')
create_subtitle(fig, grid[1, ::], 'OTH_sample_images')
create_subtitle(fig, grid[2, ::], 'PEB_sample_images')
create_subtitle(fig, grid[3, ::], 'PLM_sample_images')

plt.tight_layout(h_pad=2)
```

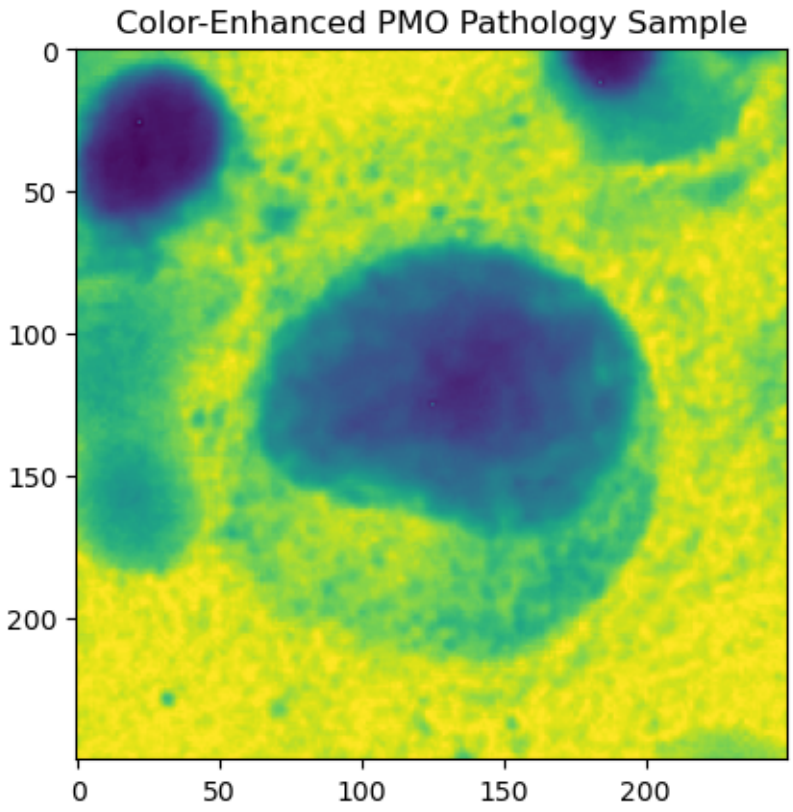


We can see by visualizing samples from each pathology that there are a lot of similarities between the differing pathologies to the naked eye, so I want to apply some pseudocolor schemes to see if we can enhance the contrast between the images.

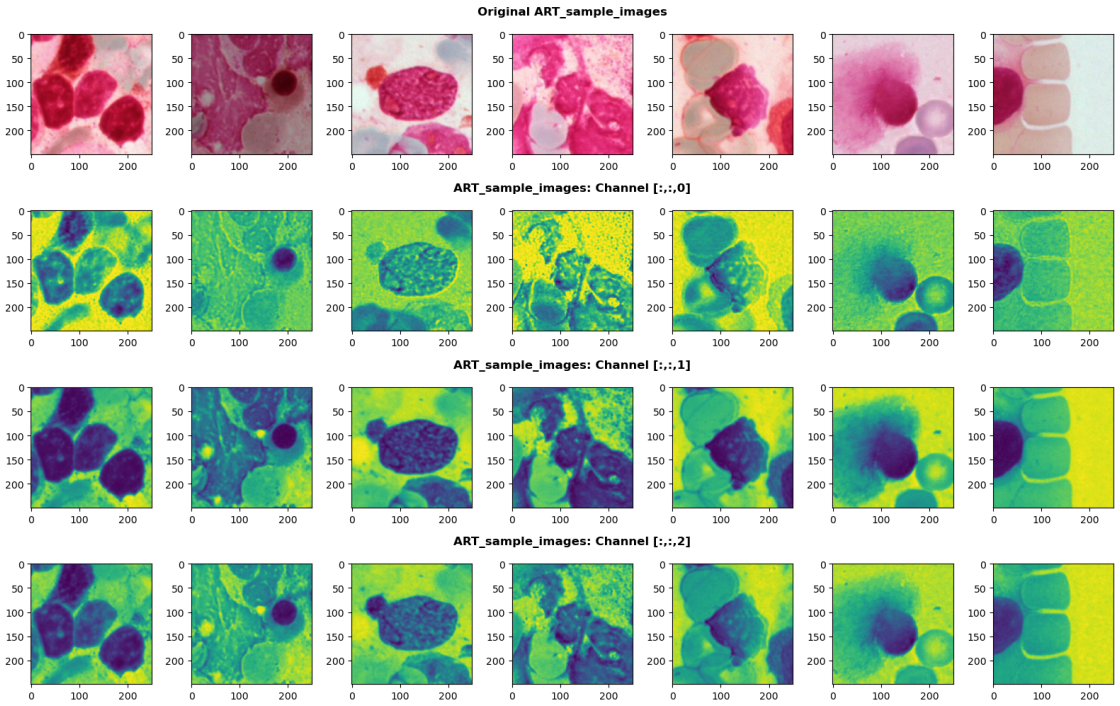
[35]:
```
#testing color enhancement on our test image
# img = PMO_sample_images[3]
cmap = plt.cm.viridis
lum_img = img#[:,:,0]

fig.set_size_inches(8, 5)
plt.title('Original Color of PMO Pathology Sample')
imgplot = plt.imshow(lum_img)
```



[36]:
```
#testing color enhancement on our test image
# img = PMO_sample_images[3]
cmap = plt.cm.viridis
lum_img = img[:,:,0]

fig.set_size_inches(8, 5)
plt.title('Color-Enhanced PMO Pathology Sample')
imgplot = plt.imshow(lum_img)
```

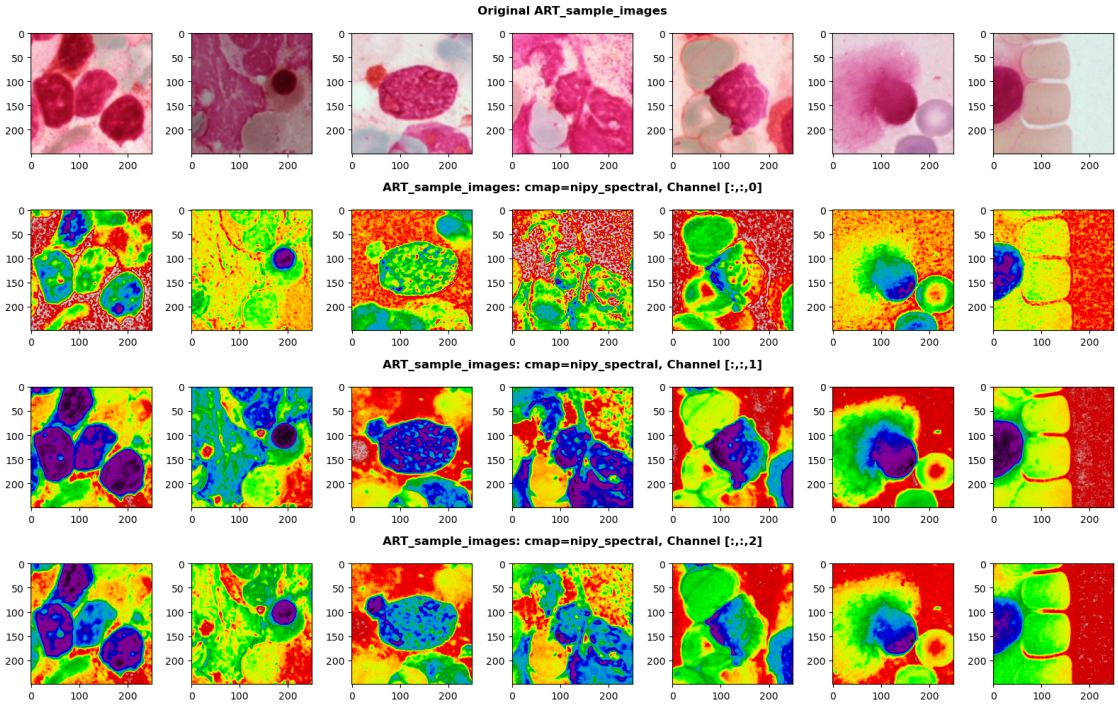


[37]:
```
#ART sample images: original images compared to manipulated RGB channels
fig, axs = plt.subplots(4, 7)
fig.set_size_inches(16, 10)

for i in range(7):
    axs[0][i].imshow(ART_sample_images[i]) #original
    axs[1][i].imshow(ART_sample_images[i][:,:,0])
    axs[2][i].imshow(ART_sample_images[i][:,:,1])
    axs[3][i].imshow(ART_sample_images[i][:,:,2])


create_subtitle(fig, grid[0, ::], 'Original ART_sample_images')
create_subtitle(fig, grid[1, ::], 'ART_sample_images: Channel [:,:,0]')
create_subtitle(fig, grid[2, ::], 'ART_sample_images: Channel [:,:,1]')
create_subtitle(fig, grid[3, ::], 'ART_sample_images: Channel [:,:,2]')

plt.tight_layout(h_pad=2)
```



[38]:
```
#ART sample images: original images compared to manipulated RGB channels
cmap = plt.cm.viridis
plt.set_cmap('nipy_spectral')

fig, axs = plt.subplots(4, 7)
fig.set_size_inches(16, 10)

for i in range(7):
    axs[0][i].imshow(ART_sample_images[i]) #original
    axs[1][i].imshow(ART_sample_images[i][:,:,0])
    axs[2][i].imshow(ART_sample_images[i][:,:,1])
    axs[3][i].imshow(ART_sample_images[i][:,:,2])


create_subtitle(fig, grid[0, ::], 'Original ART_sample_images')
create_subtitle(fig, grid[1, ::], 'ART_sample_images: cmap=nipy_spectral,␣
 ↪Channel [:,:,0]')
create_subtitle(fig, grid[2, ::], 'ART_sample_images: cmap=nipy_spectral,␣
 ↪Channel [:,:,1]')
create_subtitle(fig, grid[3, ::], 'ART_sample_images: cmap=nipy_spectral,␣
 ↪Channel [:,:,2]')

plt.tight_layout(h_pad=2)
```

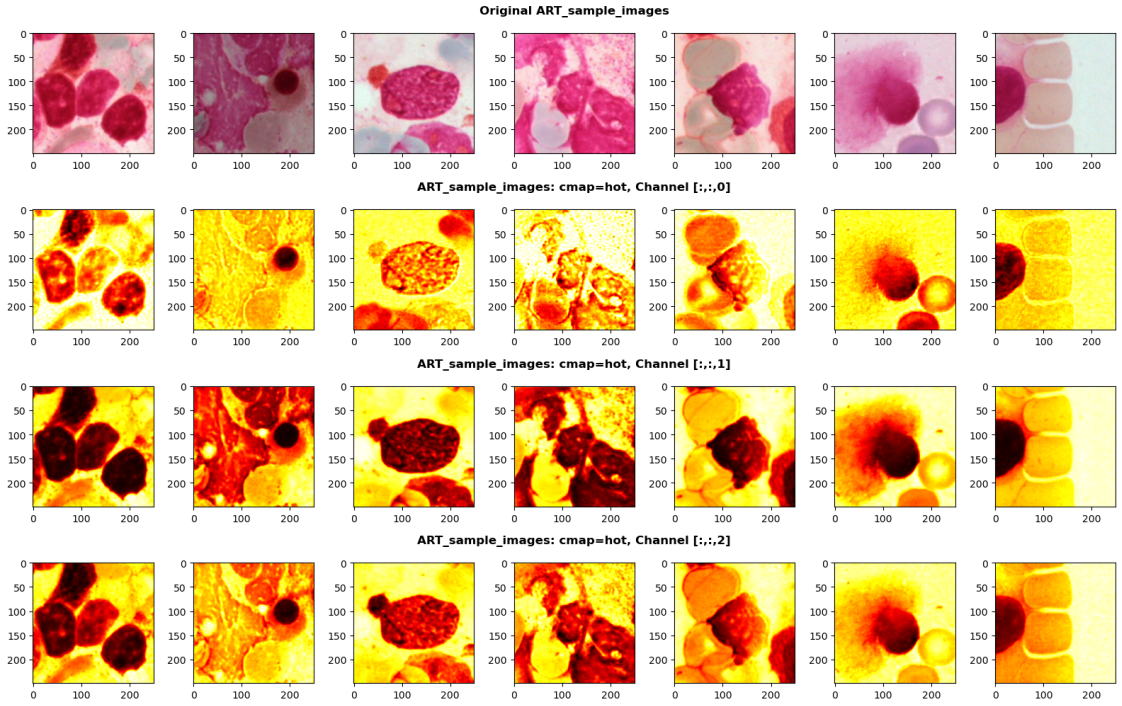
<Figure size 640x480 with 0 Axes>



[39]:
```
#ART sample images: original images compared to manipulated RGB channels
cmap = plt.cm.viridis
plt.set_cmap('hot')

fig, axs = plt.subplots(4, 7)
fig.set_size_inches(16, 10)

for i in range(7):
    axs[0][i].imshow(ART_sample_images[i]) #original
    axs[1][i].imshow(ART_sample_images[i][:,:,0])
    axs[2][i].imshow(ART_sample_images[i][:,:,1])
    axs[3][i].imshow(ART_sample_images[i][:,:,2])


create_subtitle(fig, grid[0, ::], 'Original ART_sample_images')
create_subtitle(fig, grid[1, ::], 'ART_sample_images: cmap=hot, Channel [:,:
↪,0]')
create_subtitle(fig, grid[2, ::], 'ART_sample_images: cmap=hot, Channel [:,:
↪,1]')
create_subtitle(fig, grid[3, ::], 'ART_sample_images: cmap=hot, Channel [:,:
↪,2]')
```

```
plt.tight_layout(h_pad=2)
```

```
<Figure size 640x480 with 0 Axes>
```



0.0.2 RGB Distribution

[40]:
```
# tuple to select colors of each channel line
colors = ("red", "green", "blue")

#creating the histogram plot with RBG distribution
img = PMO_sample_images[3]
plt.figure()
plt.xlim([0, 256])
for channel_id, color in enumerate(colors):
    histogram, bin_edges = np.histogram(
        img[:, :, channel_id], bins=256, range=(0, 256)
    )
    print(img[:, :, channel_id])
    plt.plot(bin_edges[0:-1], histogram, color=color)

plt.title("Color Histogram of ")
plt.xlabel("Color value")
plt.ylabel("Pixel count")
```

```
[[185 184 187 … 191 202 206]
```

```
 [183 183 186 … 195 204 206]
 [181 180 184 … 196 202 205]
 …
 [252 252 252 … 226 225 224]
 [248 250 251 … 227 231 230]
 [247 252 255 … 225 230 229]]
[[127 126 125 …  58  69  74]
 [125 125 124 …  61  70  74]
 [123 122 122 …  62  67  71]
 …
 [171 171 170 … 172 171 170]
 [170 172 173 … 175 177 176]
 [172 177 180 … 174 176 175]]
[[145 144 141 …  74  85  90]
 [143 143 140 …  79  88  91]
 [141 140 138 …  82  89  93]
 …
 [166 166 165 … 179 178 177]
 [163 165 167 … 182 184 181]
 [164 169 174 … 181 181 180]]
```

[40]: `Text(0, 0.5, 'Pixel count')`